Contextual Security Analysis (CSA)

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Firstly, a big THANK YOU! to:

- Troy Marshall
- Sean Poris
- Ben Pick
- Mike McCabe
- Rinaldi Rampen

- ALL OF YOU! 👏👏👏

Ken Johnson
@cktricky

Co-Host of Absolute AppSec
CTO of DryRun Security
BJJ Nerd

DRYRUN
.SECURITY

Common language / terminology

Dynamic Application Security Testing (DAST)	Sending requests to running web applications and then observing & analyzing the application's behavior.
Static Application Security Testing (SAST)	Analysis of the source code, in its a non-execution environment, for patterns & anti-patterns that indicate a potential security issue.
Contextual Security Analysis (CSA)	A modern, comprehensive risk assessment of software changes using a multitude of factors/data-points.
AI / LLM	Witch magic 🧙

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Outline

- A Brief History: DAST, SAST & Me
- Introducing Contextual Security Analysis
  - Current examples
  - Overview of CSA
- S.L.I.D.E.
  - Surface
  - Language/Framework
  - Intent
  - Detection
  - Environment
- LLMs / AI
  - Use cases
  - Examples

https://github.com/cktricky/csa/tree/OWASP-NoVA

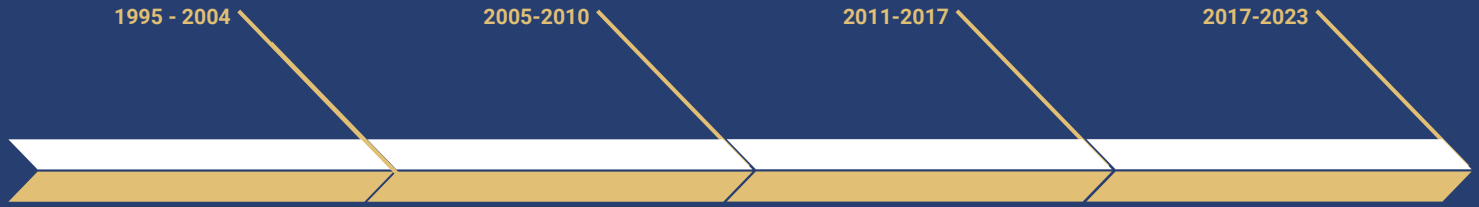
A Brief History:

SAST/DAST
&
Me

DRYRUN
.SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

A Brief History: SAST, DAST, & Me



Me: School & Navy

95 - JavaScript released
98 - SQL Injection
98 - AppScan by Sanctum
00? - Web Inspect by Spi Dynamics

Me: Pentagon / FishNet Security

02 Ounce labs
03 Fortify
06 Veracode
06 CheckMarx

Me: LivingSocial / xVisium

12 - Dependency Check
12 - A9 OWASP Category introduces
13 - Brakeman
15 - Retire.js
15 Synk launches

Me: GitHub / DryRun Security

19 Semgrep
19 CodeQL
22 Co-Pilot

IMPORTANT!!!

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Takeaways

🖥️ DAST sort of dropped off along the way…

👩‍💻Tools marketed with developers in mind

⚙️Tools still built with Security expertise required (problematic for Security Champions)

🤦‍♀️**Perfection in DAST/SAST does not exist. Our tolerance for mistakes is way higher than we'd like to admit**

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Known Gaps

Technology stacks are adopted faster than we can write tools

Scanners of today are a singular, mixed-result accuracy data point

Completely ignores the following data points:

- Previously discovered vulnerabilities
- Known dangerous portions of a codebase
- Code Quality risk markers
- Authorship completely ignored as a risk factor
- Existing Service Risk
- … The list goes on (S.L.I.D.E.)

Conclusion

Write something:

✅ Practical for people defending

✅ Harnesses existing tooling & pipeline

✅ Build on top of existing tribal knowledge

✅ Balance between adaptability/speed and perfection

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Contextual Security Analysis (CSA)

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Tell me you're doing CSA without telling me you're…

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Practical Use Cases

Awareness:

Looking for new risks in code changes using more than one factor

Variant Analysis & Regressions:

Looking for similar issues that have cropped up in the past

Remediation:

Assist developers

Chime & Monocle

SECURITY SCORE FACTORS

Docker image vulnerabilities goal: zero criticals/highs	0% of 5%
Gem security issues goal: zero criticals/highs	✅
Approved base images	✅
Brakeman enabled	✅
Bundler-audit enabled	✅
Default branch protected	✅
Github security alerts goal: zero criticals/highs	✅
Security Scan enabled	✅
CircleCI enabled	✅
CodeClimate enabled	✅
CodeClimate security issues goal: zero criticals/highs	✅
Updated within 1 month To score full marks, please make frequent updates. This reduces the number of vulnerabilities in dependencies and in application code.	✅

📚 https://medium.com/life-at-chime/monocle-how-chime-creates-a-proactive-security-engineering-culture-part-1-dedd3846127f

✅ **USES MULTIPLE RISK FACTORS** 💥

✅ Practical for people defending

✅ Harnesses existing tooling & pipeline

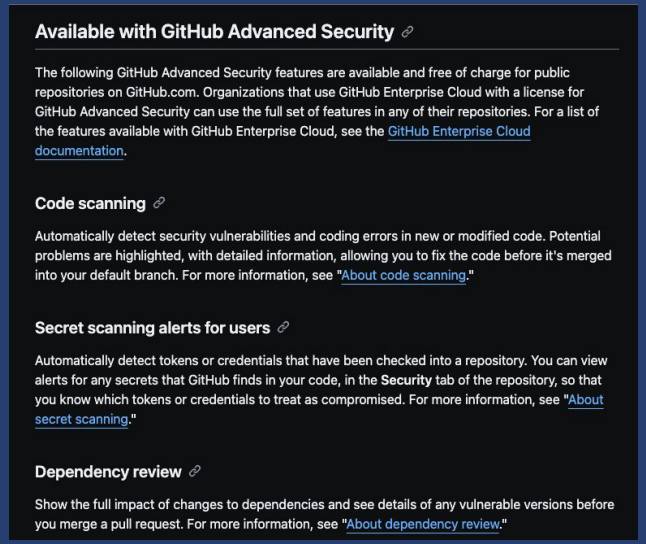
✅ Balance between adaptability/speed and perfection

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

GitHub Advanced Security (GHAS)

https://docs.github.com/en/code-security/getting-started/github-security-features#available-with-github-advanced-security



Available with GitHub Advanced Security

The following GitHub Advanced Security features are available and free of charge for public repositories on GitHub.com. Organizations that use GitHub Enterprise Cloud with a license for GitHub Advanced Security can use the full set of features in any of their repositories. For a list of the features available with GitHub Enterprise Cloud, see the GitHub Enterprise Cloud documentation.

Code scanning

Automatically detect security vulnerabilities and coding errors in new or modified code. Potential problems are highlighted, with detailed information, allowing you to fix the code before it's merged into your default branch. For more information, see "About code scanning."

Secret scanning alerts for users

Automatically detect tokens or credentials that have been checked into a repository. You can view alerts for any secrets that GitHub finds in your code, in the **Security** tab of the repository, so that you know which tokens or credentials to treat as compromised. For more information, see "About secret scanning."

Dependency review

Show the full impact of changes to dependencies and see details of any vulnerable versions before you merge a pull request. For more information, see "About dependency review."

REVEALS MULTIPLE RISK FACTORS:

- Dependency scanning
- CodeQL results
- Secret Scanning

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Practical Use Cases

Awareness:

Looking for new risks in code changes using more than one factor

Variant Analysis & Regressions:

Looking for similar issues that have cropped up in the past

Remediation:

Assist developers

Semgrep

📚https://www.jit.io/blog/semgrep-to-uncover-log4j-vulnerabilities

✅ Practical for people defending

✅ Priotizes speed (important when an incident occurs)

✅ Harnesses existing tooling & pipeline

```
root@                :/home/lahiru/SemgrepTest# semgrep --config=auto /home/lahiru/code/

 Scan Status

Scanning 15 files tracked by git with 1065 Code rules:

 Language      Rules   Files          Origin      Rules
 <multilang>      51      30          Community    1065
 java            116       2
 yaml             27       1
 dockerfile        4       1
 bash              4       1

                                                  100% 0:00:00

 1 Code Finding

    /home/lahiru/code/log4shell-vulnerable-app-main/README.md
       generic.secrets.security.detected-aws-access-key-id-value.detected-aws-access-key-id-value
          AWS Access Key ID Value detected. This is a sensitive credential and should not be hardcoded
          here. Instead, read this value from an environment variable or keep it in a separate,
          private file.
          Details: https://sg.run/GeD1

            22┆  ...
AKIAIWNJYAX4CSVEH53A%2F20211211%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20211211T031401Z&X-Amz-Expires
=300&X-Amz-Signature=140e57e1827c6f42275aa ...
           [shortened a long line from output, adjust with --max-chars-per-line]

 Scan Summary

Ran 1065 rules on 15 files: 1 finding.
root@                :/home/lahiru/SemgrepTest#
```

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Practical Use Cases

Awareness:

Looking for new risks in code changes using more than one factor

Variant Analysis & Regressions:

Looking for similar issues that have cropped up in the past

Remediation:

Assist developers

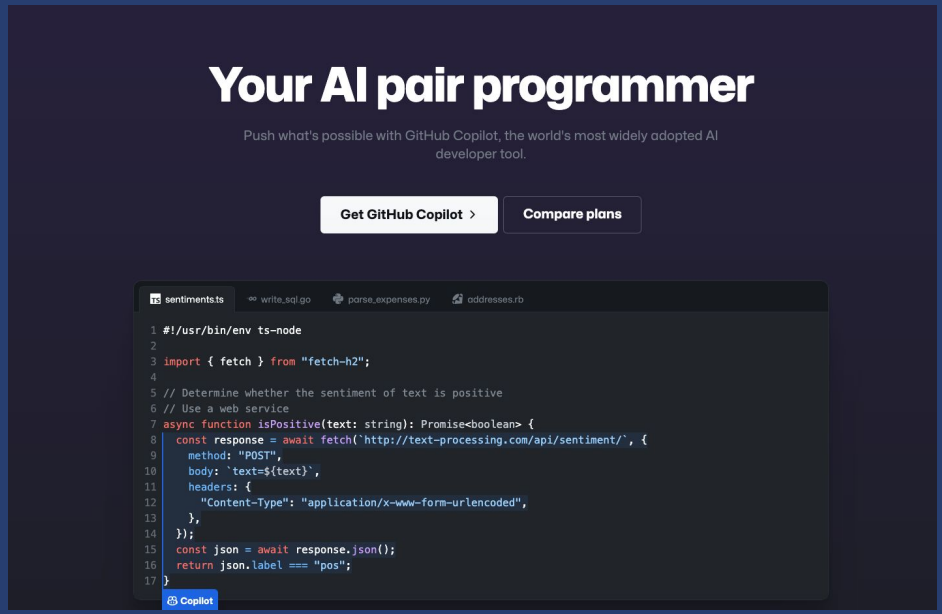
Co-Pilot

📚https://github.com/features/copilot

✅ Practical for people defending

✅ Prioritizes speed over perfection

✅ Harnesses existing tooling & pipeline

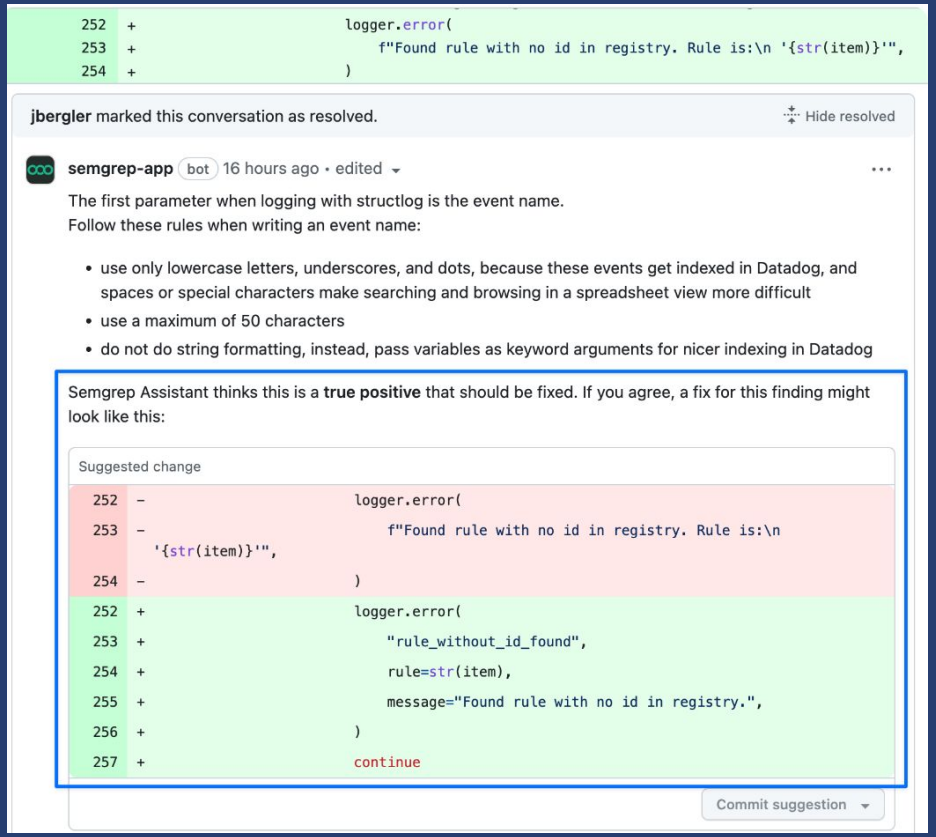


DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Semgrep Assistant

https://semgrep.dev/products/semgrep-code/assistant/



DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Key concepts 💡

Ideally, we merge these strategies together

There are still signals for risk that we're not yet tapping into

We need tooling to keep pace with internal technology adoption

Scaling knowledge is key

Overview

https://github.com/cktricky/csa/tree/OWASP-NoVA

Context > Control

https://github.com/cktricky/csa/tree/OWASP-NoVA

Control

(S,D,I)AST embodies this:

😭Limited data points

😭Enforcement of rules

😭Blocking checkpoints

😭Pattern matching

Context

Next Gen:

😍Combines many data points

😍Warnings & guidance over enforcement

😍 Remediation assistance

😍 Risk-focused

Download Link: https://www.dryrun.security/resources/csa-guide

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

CSA - SLIDE □

Surface	How the surface of the application changes
Language	Language and framework that the application is written in
Intent	Evaluates the person making the change, both in their patterns and their purpose
Detection	Tooling in place to detect vulnerabilities and security issues
Environment	Purpose of the application and service in the organization

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

SLIDE - Surface

⚠️Sensitive Code Paths

Depending on your language and framework, this could be configuration files, middleware, controllers, or how your application and service handles auth.

🚧 HTTP Routes

Routes refers to an exposed endpoint on an application that will take user-supplied information and perform an action. Expansion or contractions of these routes in an application inherently change the risk profile of the application.

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

SLIDE - Language

💬Source Language

This weighs the language used (ex: Golang, Rust, PHP, etc.) as each has their own security issues and known vulnerabilities.

🧱Web Framework

Each web framework contains their own unique variations of sensitive areas, configurations, and components. Framework nuances are subtle and highly specific security issues that are akin to “footguns” 🦶🔫.

SLIDE - Intent

🛰️ Authorship frequency	If the author rarely commits or has never committed to this repo this indicates the author may be unfamiliar with the development standards of the code base.
□Author is not an owner	Code changes to a portion of code in which the author is not an owner or on a team that owns this portion of the code.

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

SLIDE - Detection

🤐 Secrets

Secret scanning technology has enabled the detection of sensitive keys when pushed in source code. This is a major indicator of risk in a code change.

👩‍🔬 Testing

Consuming (S|D)AST, dependency, docker image, etc. scanner tool output. Past bug bounty submissions as a historical reference to known dangerous anti-patterns. Similar concept with vulnerabilities Identified by internal/external security contributors.

https://github.com/cktricky/csa/tree/OWASP-NoVA

SLIDE - Environment

💪 Change protection

Does the application utilize branch protection features or other testing-relevant gating mechanisms to ensure evaluation of new code changes?

👩‍💼 Business Risk

Critical factors include the level of risk a service poses through its business function, the type of data it processes/stores/transmits, and its overall importance to the business.

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

!(Deterministic vs Probabilistic)

https://github.com/cktricky/csa/tree/OWASP-NoVA

Surprise… it's both! 🎉

Probabilistic	Use to gather context (ie - is this touching authentication?) Use to provide (anti)pattern analysis 🧠 (ie - does this match the pattern for a previously known 🐛)
Deterministic	Also can be used to gather context (ie - Write code that checks if this is a first time repo author) Specific things that produce exact matches (ie - Write code / Semgrep pattern)
🏆**USE THE OUTPUT OF BOTH TO REVEAL RISK** 🏆	

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

AI & You

DRYRUN
.SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Personal recommendations

🔬🧪Start experimentation with Baseplate or something similar

📖 Read this article:

https://engineering.peerislands.io/extending-openai-gpt-4-using-langchain-and-pinecone-for-q-a-over-your-own-content-using-1f3e9dc10e91

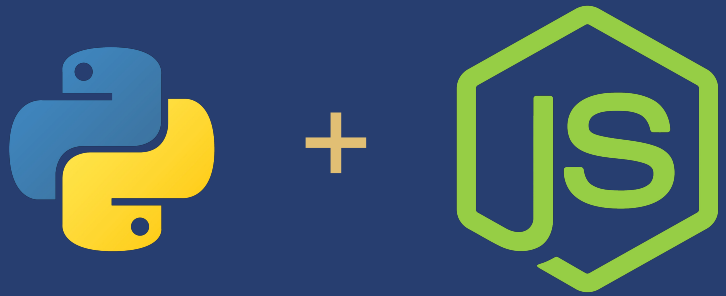
📖 Read this book:

https://www.amazon.com/What-ChatGPT-Doing-Does-Work/dp/1579550819

Langchain

Modular Components

- LLMs
- Compressors
- Retrievers
- Prompts
- Chains
- … and way more



My AI learnings…

Contextual Compression + Documents, Chains, LLMs, Prompts… they are all highly configurable and can be very accurate

Created the following:

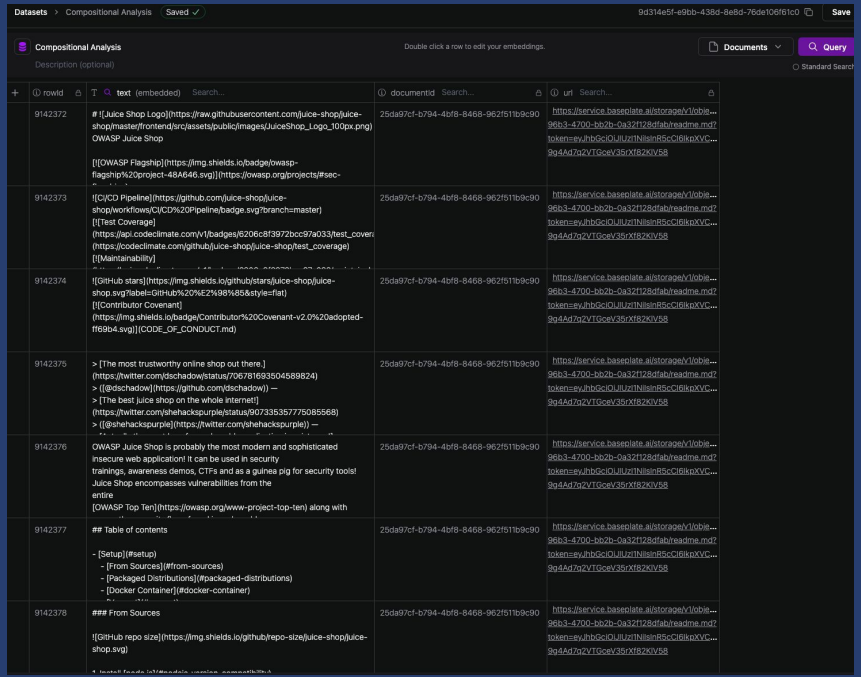
- AI Chat bot assistant that can dig through your organization's secure code docs to answer questions
- Analysis agent that provides composition of the application (framework, language, security centric libraries, datastore, etc.)
- Analysis agent that looks for known (in your app's specific code base) security bug patterns

https://github.com/cktricky/csa/tree/OWASP-NoVA

Baseplate - Create Dataset

Dataset:

Juice Shop's **README.md** and **package.json** files



DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Baseplate - Create App

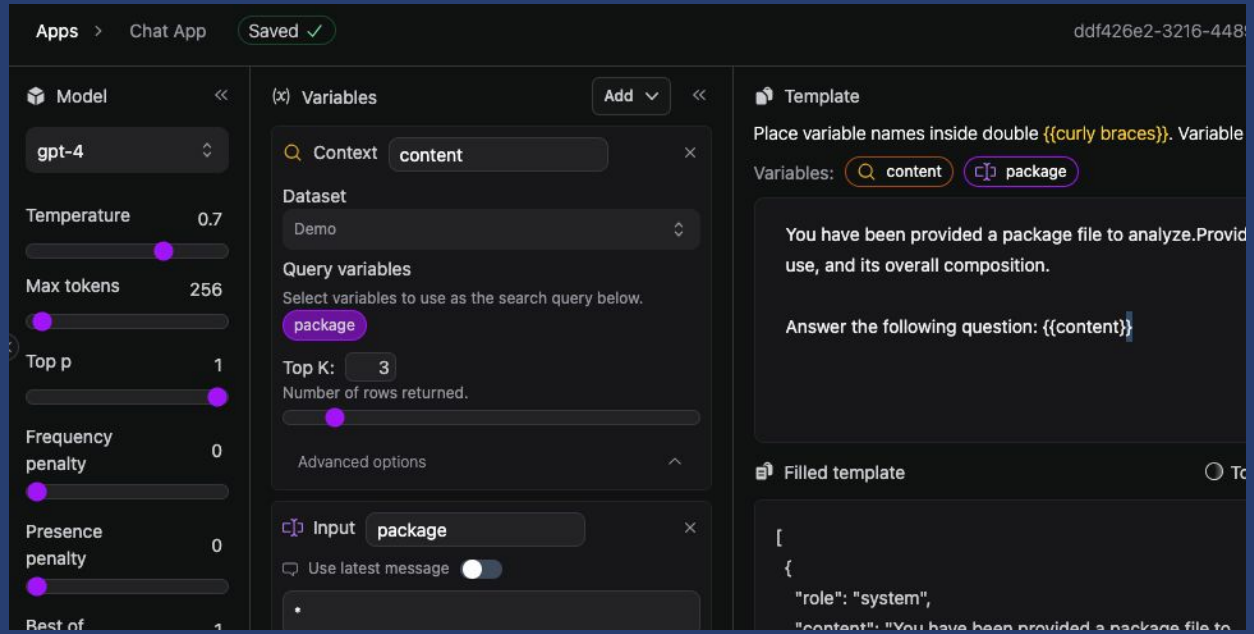
Provide the following information:

Template

Context (point to a dataset)

Input Variables (Optional)

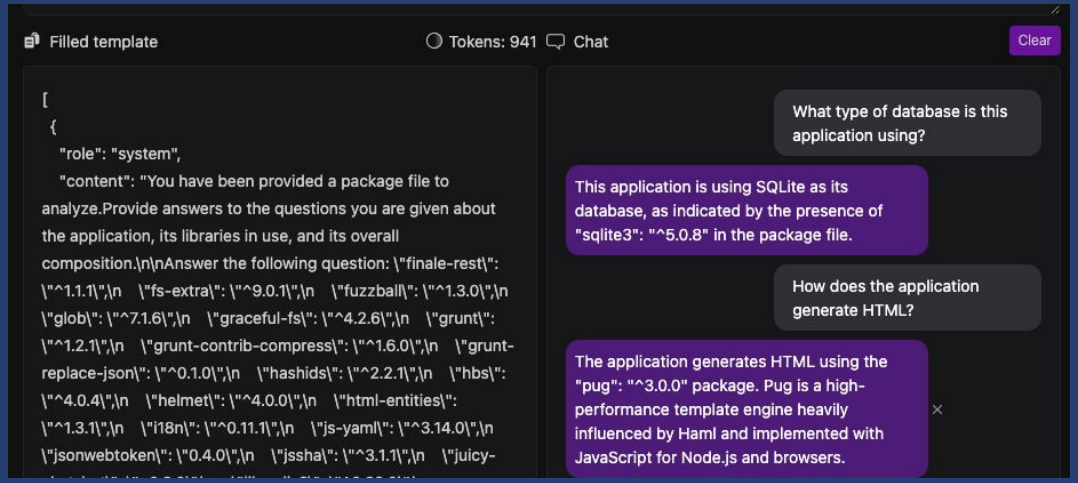
Start asking questions…

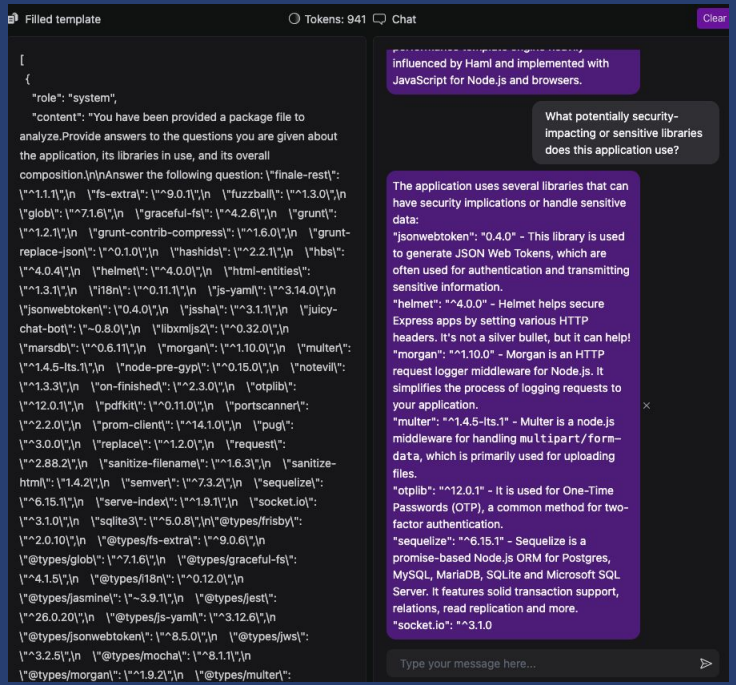


https://github.com/cktricky/csa/tree/OWASP-NoVA

DRYRUN .SECURITY

Baseplate - Ask questions





DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Baseplate Wrap-up

Good for:

- Testing
- Proof of concept
- Creating a chat agent quickly
- Learning!

Less good for:

- Intense customization
- More than chat, though not impossible by any stretch

Baseplate

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Customization using Langchain

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Langchain “Chain” ⛓

- Pinecone vector store
- OpenAI embeddings
- Prompts (System, Human, Chat)

```
# Convert to an OpenAI Embedding model
pinecone_vector_store = Pinecone.from_existing_index(index_name=pinecone_index,embeddir
print("FINISHED CREATING EMBEDDING MODEL")

# Create system prompt template text
template = '''
Answer questions related to software security.
Supply all answers in a Markdown format suitable for placing inside GitHub issue commer

If you don't know the answer, just say that "I do not yet know the answer to your quest
please ask your security team", don't try to make up an answer.

{summaries}
'''

# Convert text to system prompt
system_message_prompt = SystemMessagePromptTemplate.from_template(template)

# Take the value of question and place it within the human prompt message
human_template = "{question}"
human_message_prompt = HumanMessagePromptTemplate.from_template(human_template)

# Combine our system and human prompts to form our "Chat Message Prompt"
prompt = ChatPromptTemplate.from_messages([system_message_prompt, human_message_prompt]

# Create arguments that will be pushed into our chain
chain_type_kwargs = {"prompt": prompt}
```

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

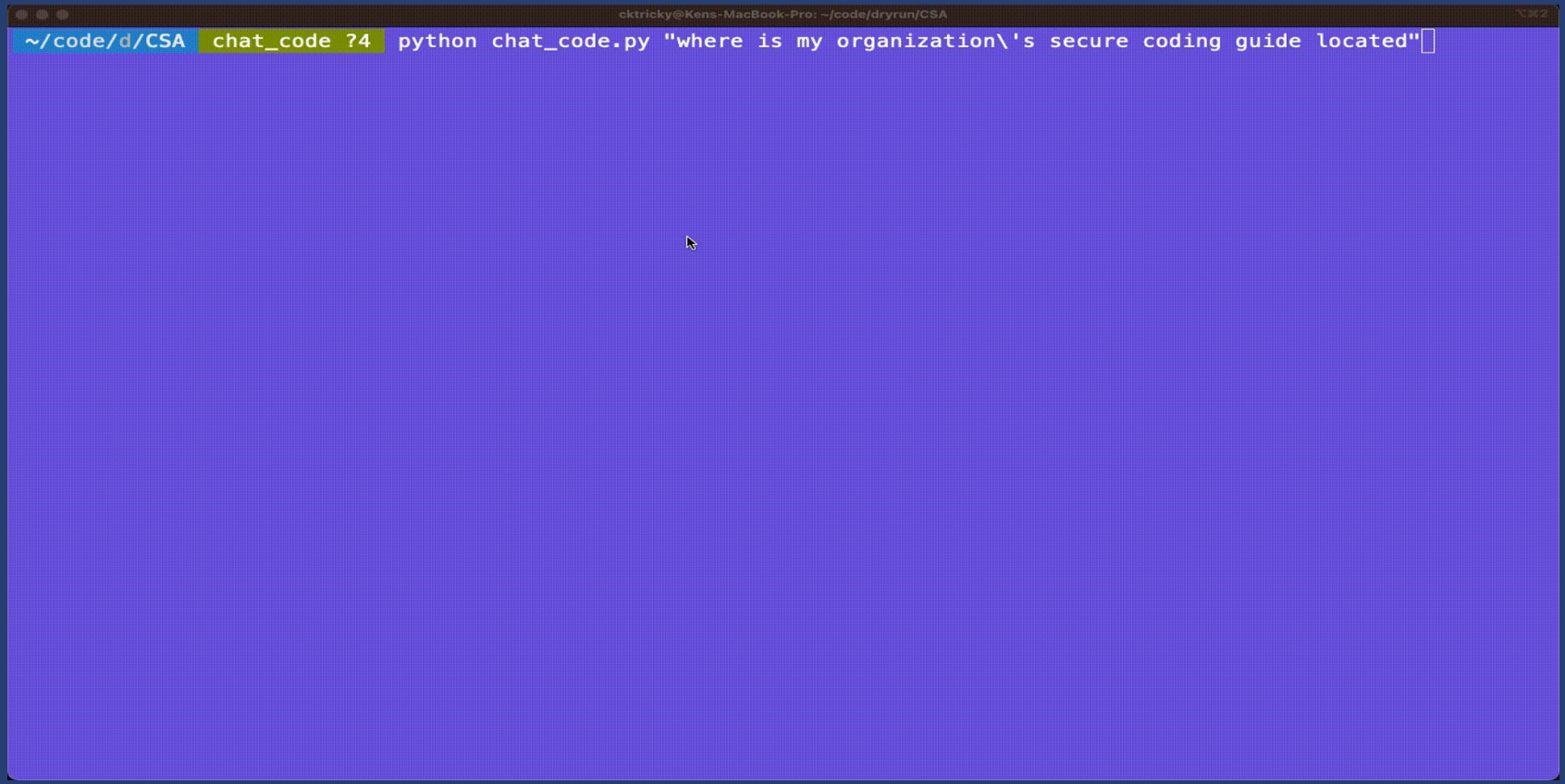
Langchain “Chain” ⛓

- OpenAI LLM
- Contextual Compression
- Retriever

```
# Choose our openAI chat model and assign a temperature.
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0.7, max_tokens=256)

# We need to extract and compress the relevant documents so we're giving the
# OpenAI llm as an option here
compressor = LLMChainExtractor.from_llm(llm)

# This has our compressor and vector db options. It is pulling only the relevant text f
# document retrieved based on a user's query rather than the entire document contents.
compression_retriever = ContextualCompressionRetriever(base_compressor=compressor, base

# This performs retrieval of our answers based on our data and context
# and chains together our LLM, our compression retriever, and prompt.
chain = RetrievalQAWithSourcesChain.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=compression_retriever,
    return_source_documents=True,
    chain_type_kwargs=chain_type_kwargs,
    verbose=True
)

response = chain(question)
reply = response["answer"]
print(f"Reply: {reply}")
```

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA



DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Final thoughts on building your own AI powered tooling

🔮Know what you're trying to use it for

🚧Integrate into existing tooling and processes

📊 Harness documentation, tribal knowledge, open source security info, etc.

🥊Combine results from many sources to represent an accurate picture of risk

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Conclusion

DRYRUN .SECURITY

https://github.com/cktricky/csa/tree/OWASP-NoVA

Things are changing… they must

- 💡 Too many technology choices to continue testing in isolation using a single data point to determine risk
- 💡 The power is in your hands to build your own highly customized tooling
- 💡 Meet developers where they live
- 💡 AI / LLM can be a for multiplier; is NOT a silver bullet

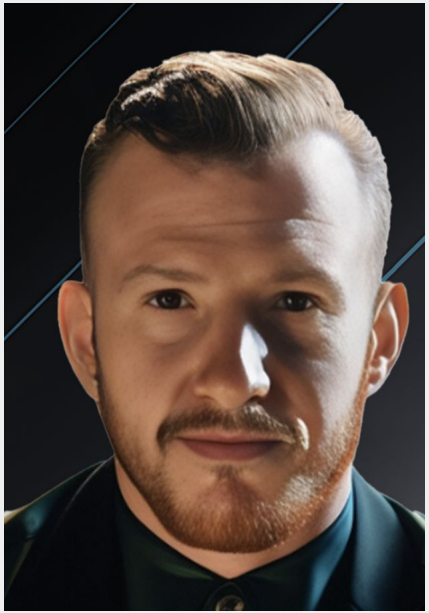
https://github.com/cktricky/csa/tree/OWASP-NoVA

Contact Info

@cktricky

ken@dryrun.security

youtube.com/@AbsoluteAppSec



DRYRUN
.SECURITY